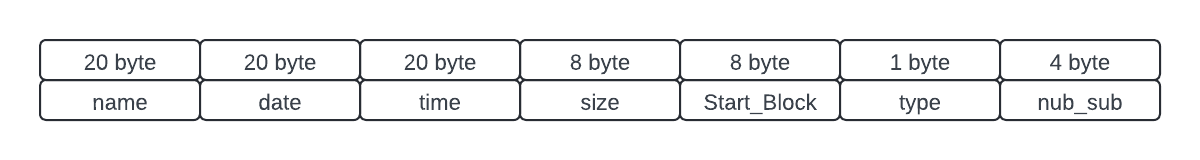
**Operating System  
HW\_2**

**Ziya Kadir TOKLUOGLU**

**Introduction:**  
This project involves the implementation of a simplified CD-ROM file system.The main objective was to design a file system that efficiently manages storage space while minimizing internal fragmentation. The file system was structured in three main stages: first, the superblock was created, which holds essential metadata about the file system. Next, directory entries were organized after all files were organized. This approach ensures that the file entries are stored contiguously, reducing wasted space within the blocks. Although the project focuses on fundamental concepts, it effectively demonstrates key principles of file system design, such as the importance of managing fragmentation and ensuring efficient storage allocation.

**Detail of File\_Entry Structure**



typedef struct file\_entry{

char name[20]; // 20 bytes for name

char date[20]; // 10 bytes for date in YYYY-MM-DD

char time[20]; // 10 bytes for time in HH:MM:SS

int64\_t size; // 8 byte for file size

int64\_t start\_block; // 8 byte for start block

char type;

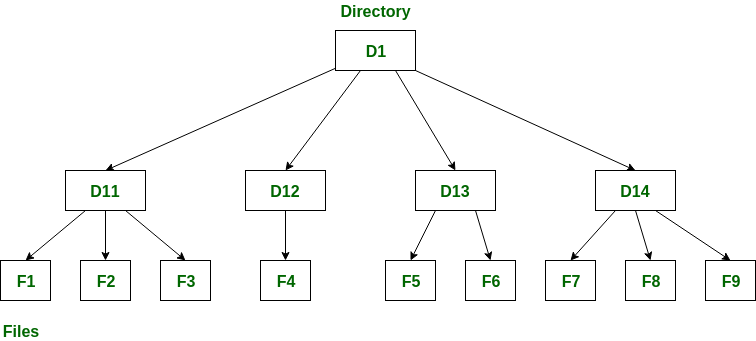
int number\_sub\_files = 1; // 1 byte for type (0 for file, 1 for directory)

} file\_entry;

We hold necessary information which are mentioned in the pdf

The size can be changed according to your preferences.

**Holding File\_system as a TREE**



Two classes are written for implementing this tree one of them is File and the other one is Directory.   
Directory is derived by File class because directories are defined as special File.

**Class File;**

class File {

public:

File(const char name[20] = "", int64\_t size = 0, time\_t modTime = 0, File\* parent = nullptr);

virtual ~File(); // Virtual to allow inheritance

static int entry\_size; // Size of each entry in the file system

const char\* get\_name() const;

const char\* get\_date() const;

const char\* get\_time() const;

int64\_t get\_start\_block() const;

int64\_t get\_size() const;

File\* get\_parent() const;

virtual char get\_type() const;

file\_entry get\_entry() const;

void set\_entry(file\_entry entry);

virtual void add\_file(File\* file);

void set\_name(const char name[20]);

void set\_date(const char date[10]);

void set\_time(const char time[10]);

void set\_start\_block(int64\_t start\_block);

void set\_size(int64\_t size);

void set\_type(char type);

void set\_parent(File\* parent);

virtual int get\_number\_of\_subfiles() const;

virtual void set\_number\_of\_subfiles(int number\_of\_subfiles);

protected:

file\_entry entry;

File\* parent; // Pointer to parent file

void formatDateTime(time\_t rawTime);

};

There is no need to explain all getter and setter functions; their functionalities are clear.

We focus on

virtual void add\_file(File\* file);

This is the crucial function for creating a tree structure. We add all files to our tree and after we print after the superblock in this way we hold all file entries together in the blocks instead allocating blocks for each file in different blocks. In this way we really prevent ***INTERNAL FRAGMENTATION.***

**Directory class:**

#ifndef DIRECTORY\_H

#define DIRECTORY\_H

#include "File.h"

#include <vector>

#include <ctime>

using namespace std;

class Directory : public File {

public:

Directory(const char name[20], int64\_t size = 0, time\_t modTime = 0, File\* parent = nullptr);

virtual ~Directory();

virtual void add\_file(File\* file) override; // Override to add file-specific functionality

const vector<File\*>& getFiles() const;

int getNumFiles() const;

int get\_number\_of\_subfiles() const;

void set\_number\_of\_subfiles(int number\_of\_subfiles);

vector<File\*>& getFiles();

File\* operator[](int index) const;

private:

vector<File\*> files; // Using pointers to manage file objects

};

#endif // DIRECTORY\_H

as you can see we hold the sub files and directories in a vector.  
While adding a file to TREE , all necessary information is modified because we fill out bottom to top with a recursive function. It will be mentioned below.

void Directory::add\_file(File\* file) {

if (file) {

files.push\_back(file);

file->set\_parent(this); // Set this directory as the parent

this->set\_size(get\_size() + file->get\_size());

this->set\_number\_of\_subfiles(this->get\_number\_of\_subfiles() + file->get\_number\_of\_subfiles());

}

}

**Creating a TREE STRUCTURE**

void takeDirectory(const string& dirPath, Directory& parentDirectory) {

DIR\* dir = opendir(dirPath.c\_str());

if (dir == nullptr) {

cerr << "Failed to open directory: " << dirPath << endl;

return;

}

dirent\* entry;

while ((entry = readdir(dir)) != nullptr) {

string entryName = entry->d\_name;

if (entryName == "." || entryName == "..") continue; // Skip . and .. entries

string fullPath = dirPath + "/" + entryName;

struct stat entryStat;

if (stat(fullPath.c\_str(), &entryStat) == -1) {

cerr << "Failed to get stat for: " << fullPath << endl;

continue;

}

if (entryName.size() >= 20){

cerr << "File name is too long: " << entryName << endl;

continue;

}

if (S\_ISDIR(entryStat.st\_mode)) { // If the entry is a directory

Directory\* newDir = new Directory(entryName.c\_str(), 0, entryStat.st\_mtime);

takeDirectory(fullPath, \*newDir);

parentDirectory.add\_file(newDir);

} else if (S\_ISREG(entryStat.st\_mode)) { // If the entry is a file

File\* newFile = new File(entryName.c\_str(), entryStat.st\_size, entryStat.st\_mtime);

parentDirectory.add\_file(newFile);

}

}

closedir(dir);

}

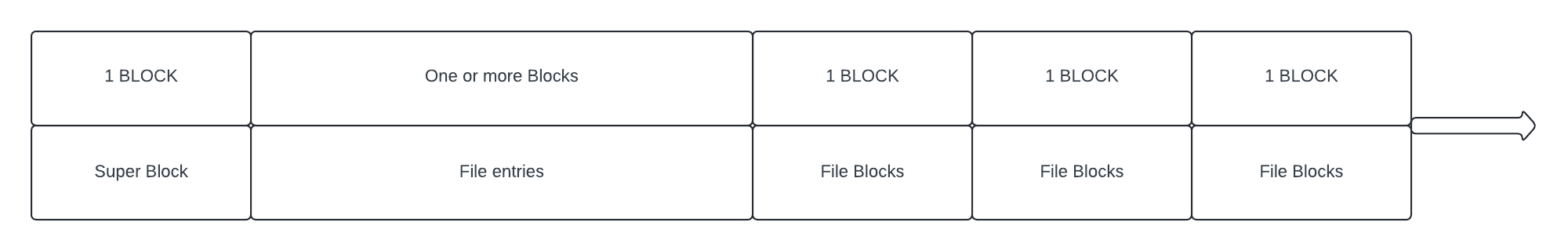
We use necessary libraries to reach the file structure of my computer. After reaching we create our TREE structure using add\_file(). The main point is:

takeDirectory(fullPath, \*newDir);

parentDirectory.add\_file(newDir);

Using this order we build tree bottom instead of top to bottom, This helps us to hold the sub\_directory number and each file\_size which includes sub files’ sizes.

**CD\_ROM Structure:**



As we mentioned, our structure tries to keep all the same type in the same blocks which pretends to pretend **INTERNAL FRAGMENTATION.** Also all blocks are in the same size which also pretends **EXTERNAL FRAGMENTATION.**

A screenshot of a computer program

Description automatically generated**READ CD\_ROM**

The most important point is read super\_block and File\_entries Block. In this way we can rebuilt out file TREE structure and reach every file and control every file we want.

A screen shot of a computer program

Description automatically generated**DECIDE Which Operation**

We decide Operation with spesific function after we decide operation and execute and list necesarry outputs:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedAlso we use these kind of operation in the write program:

I dont use chatgpt to generating code but I use chatgpt as a modifier to syntax and and understanding the topic with it’s help.